

# **Unidad III: Programación orientada a objetos intermedia**

Lic. Ronaldo Armando Canizales Turcios

Departamento de Electrónica e Informática, UCA

Ciclo virtual 01-2021

# Agenda

- 1 Introducción al manejo de Excepciones
- 2 Capturar excepciones: try-catch
- 3 Lanzar excepciones: throw
- 4 La cláusula finally
- 5 Excepciones personalizadas
- 6 Anuncios

# 1. Introducción al manejo de Excepciones

## No existe el programa perfecto

En la práctica, los programas pueden fallar por múltiples razones, a veces por descuido del programador y otras veces por mal input del usuario.

Hoy veremos una de las herramientas más utilizadas para poder enfrentar errores de todo tipo: el manejo de **excepciones**.

### excepcional

1. **adj.** Que constituye excepción de la regla común.
2. **adj.** Que se aparta de lo ordinario, o que ocurre rara vez.

*Real Academia Española © Todos los derechos reservados*

Su significado<sup>1</sup> nos indica que normalmente no deberían suceder, pero un buen programador siempre debe estar listo para enfrentarlas y hacer sus app's "a prueba de balas".

---

<sup>1</sup>Diccionario de la lengua española: [dle.rae.es/excepcional](https://dle.rae.es/excepcional)

# 1. Introducción al manejo de Excepciones

La ventaja principal del manejo de excepciones es que **automatiza** gran parte del código de manejo de errores que previamente (en lenguajes más antiguos) debía ingresarse manualmente, cada vez que se llama a un método. En cualquier programa grande este enfoque es tedioso y propenso a errores.

Por ejemplo, supongamos un **método** que recibe un código postal y devuelve el nombre del ayuntamiento al cual pertenece. En el caso normal de que el código postal sea correcto, el método devolvería el nombre del ayuntamiento. En el caso de que el código postal que se le haya pasado no exista o su formato sea incorrecto, el método lo notificaría **lanzando una excepción**.

# 1. Introducción al manejo de Excepciones

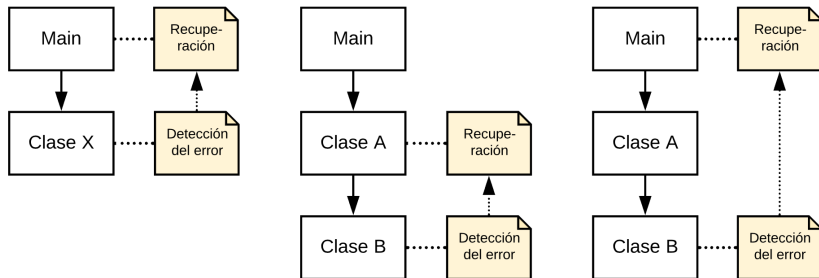
Existen dos etapas en el manejo de los errores:

- Detección del error.
- Recuperación.

Lanzar una excepción

Capturarla y tomar alguna medida

El manejo de errores en la programación orientada a objetos presenta un gran **desafío**: usualmente el lugar donde sucede la **detección** del error es distinto del lugar donde ocurre la toma de decisión para la **recuperación**.



# 1. Introducción al manejo de Excepciones

## Por ejemplo, imagine que:

- ❶ El usuario ingresa una cantidad negativa (en el main).
- ❷ Luego dicho valor se envía como argumento a un método de la **clase X**.
- ❸ Dicha cantidad negativa ocasiona un error (se detecta el error).
- ❹ ¿Qué debería hacer la **clase X**?
  - ¿Debería notificar al usuario y pedirle otro valor? (oops, esto debería hacerse en el main).
  - ¿Debería abortarse la operación? (esa decisión le pertenece al usuario).

La verdad es que, el método de la clase X no posee la información suficiente como para tomar una decisión. Lo correcto sería abandonar el método inmediatamente se detecte el error y notificar al main. Será el main el encargado de **recuperarse** del error, es decir **tomar una decisión**.

# 1. Introducción al manejo de Excepciones

El manejo de excepciones en C# se gestiona a través de cuatro palabras clave: **try**, **catch**, **throw** y **finally**. Forman un subsistema interrelacionado en el que el uso de uno implica el uso de otro.

Las instrucciones que se desea supervisar (porque podrían lanzar excepciones) están contenidas dentro de un bloque **try**. Si se produce una excepción dentro del bloque **try** se puede atrapar dicha excepción usando **catch** y manejarlo de una manera racional (recuperación).

Para lanzar manualmente una excepción, use la palabra clave **throw**. Cualquier código que debe ejecutarse al salir de un bloque **try** se coloca en un bloque **finally**.



# 1. Introducción al manejo de Excepciones

## Resumen:

- **try:** se utiliza para encerrar un conjunto de instrucciones de las que se desconfía en un bloque. El bloque try debe ir seguido de al menos un bloque **catch**, es decir, no se puede usar un try él solo.
- **catch:** se utiliza para capturar y manejar excepciones. Se colocan después de los bloques **try**. Pueden ir seguidos de un bloque **finally**.
- **finally:** se coloca código que se ejecutará **después** de un bloque try, independientemente suceda una excepción o no. Su uso es **opcional**.
- **throw:** se utiliza para lanzar excepciones de forma “manual”.



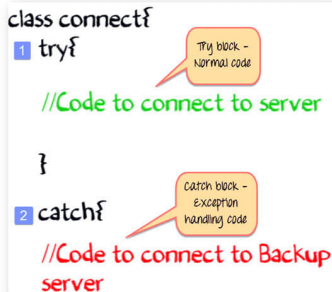
## 2. Capturar excepciones: try-catch

Todas las excepciones deben ser capturadas **en algún lugar** del programa. Si una excepción no es capturada, entonces se muestra un mensaje de error y el programa completo finaliza abruptamente.

Para eso existe la sentencia **try-catch**, que está conformada por un bloque **try** donde se colocan las instrucciones que *podrían* lanzar excepciones y uno o más bloques **catch**, donde se capturan distintos tipos de excepciones.

Estas palabras clave trabajan juntas, no puedes atrapar (catch) sin intentarlo (try)

```
class connect{  
  1 try{  
    //Code to connect to server  
  }  
  2 catch{  
    //Code to connect to Backup  
    server  
  }  
}
```



The diagram illustrates the try-catch structure. A callout bubble points to the try block, stating 'Try block - Normal code'. Another callout bubble points to the catch block, stating 'Catch block - exception handling code'.



## 2. Capturar excepciones: try-catch

### Varios bloques catch para un mismo bloque try

Es posible que las instrucciones contenidas dentro del bloque **try** generen distintos tipos de excepciones, entonces se recomienda implementar un bloque **catch** por **cada tipo** de excepción que *podría* ser generada, **de la más específica** (sub-clases) hasta la **más general** (la súper-clase Exception).



- Si el denominador es cero, se generará una DivideByZeroException.
- Si el índice está fuera de los límites del arreglo, se generará una IndexOutOfRangeException.
- Las clases más generales (súper-clases) deben colocarse al final, sino absorberían los demás catch's (por concepto de polimorfismo).

### 3. Lanzar excepciones: throw

Los ejemplos anteriores han estado capturando excepciones generadas automáticamente. Sin embargo, es posible lanzar manualmente una excepción utilizando la instrucción `throw`.

Cuando un método detecte una situación problemática, deberá de **lanzar** (`throw`) una excepción apropiada. Esto implica que:

- El método finaliza antes de tiempo, es decir aborta su ejecución.
- Devuelve un objeto de la clase `Exception` (o de alguna clase hija).
- Será trabajo de **quien mandó a llamar** a dicho método el capturar la excepción.

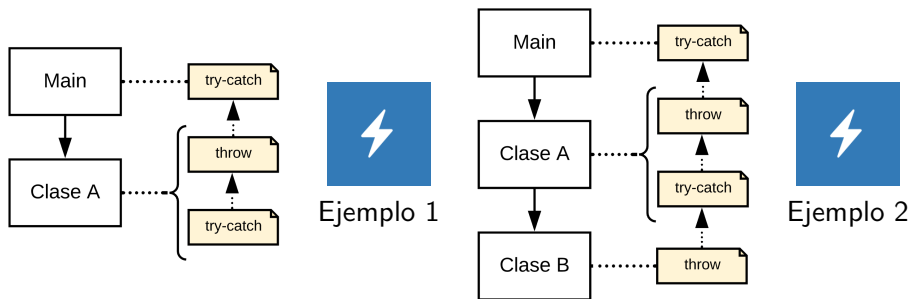
```
public static void validate(int age) {  
    if(age<18)  
        throw new ArithmeticException("Menor de edad");  
    System.out.println("Bienvenido, usted sí puede votar!");  
}
```



### 3. Lanzar excepciones: throw

#### Re-lanzar una excepción

Una excepción capturada por una declaración `catch` se puede volver a lanzar para que pueda ser capturada por un `catch` externo.



Cuando se re-lanza una excepción, no se volverá a capturar por el mismo bloque **catch** (método actual). Se propagará al siguiente bloque **try-catch** (el que hizo la llamada al método donde se originó el error).

## 4. La cláusula finally

Algunas veces querrá definir un bloque de código que se ejecutará cuando finalice un try-catch. Por ejemplo, una excepción puede causar un error que finaliza el método actual, causando su devolución prematura. Sin embargo, ese método puede haber abierto un archivo o una conexión de red que debe cerrarse independientemente si se ha producido una excepción o no. Para estos casos existe una cláusula llamada **finally**.

El bloque finally se ejecutará **siempre** que se abandone un bloque try-catch, sin importar las condiciones que lo causen. Es decir, si el bloque try finaliza normalmente (se ignoran los bloques **catch**) o debido a una excepción, el último código ejecutado es el definido por finally.

## 4. La cláusula finally

### Si se lanza alguna excepción en las instrucciones del bloque TRY

- Si existe alguna cláusula CATCH competente, entonces se ejecuta primero dicha cláusula y después se ejecutan las instrucciones contenidas en la cláusula **finally**.
- Si NO existe alguna cláusula CATCH competente, entonces primero se ejecutan las instrucciones contenidas en la cláusula **finally** y luego se aborta el método, lanzando la excepción a quien sea que haya llamado a dicho método (si no hay nadie más, entonces el programa entero termina abruptamente).



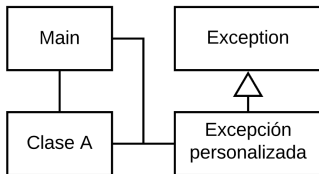
Ejemplo

## 5. Excepciones personalizadas

### Declaración de nuevas excepciones

C# cuenta con muchas excepciones predefinidas para determinadas situaciones, como por ejemplo **IOException** para errores producidos en operaciones de entrada/salida, como es el caso de la lectura de ficheros.

Para nuestros propios programas, a veces es útil que creamos nuestra excepción a medida. Para ello, tenemos que declarar una nueva clase que herede de **Exception** o de alguna otra.





## 5. Excepciones personalizadas

### Ejemplo de excepción personalizada

```
public class BadPostCodeException : Exception {  
    public BadPostCodeException() : base() {  
        Constructor clase padre (Exception)  
    }  
    public Bad...tion(string message) : base(message) {  
        Constructor con parámetro  
    }  
}
```



Ejemplo 1



Ejemplo 2

Normalmente, no necesitaremos declarar nuevos atributos ni métodos, solamente los constructores.

## 6. Anuncios

### Consejo

Lanzar excepciones sólo en casos excepcionales. Se recomienda utilizar racionalmente las excepciones, es decir, lanzarlas cuando suceda algo *inesperado*. Se debe evitar usar las excepciones como una *instrucción break con esteroides*, no utilizarlas para salir de un bucle anidado o una función recursiva (eso sería considerado un abuso del mecanismo de las excepciones).

### Comienza Meta II

Favor llenar el Google Forms que se encuentra en el Moodle.

- Todos, tanto si desean continuar con la misma pareja en el Clan.
- Aquellos clanes que deseen re-estructurarse.

### Evaluaciones Meta I

Se estima tener las notas para finales de la presente semana.

# **Unidad III: Programación orientada a objetos intermedia**

Lic. Ronaldo Armando Canizales Turcios

Departamento de Electrónica e Informática, UCA

Ciclo virtual 01-2021